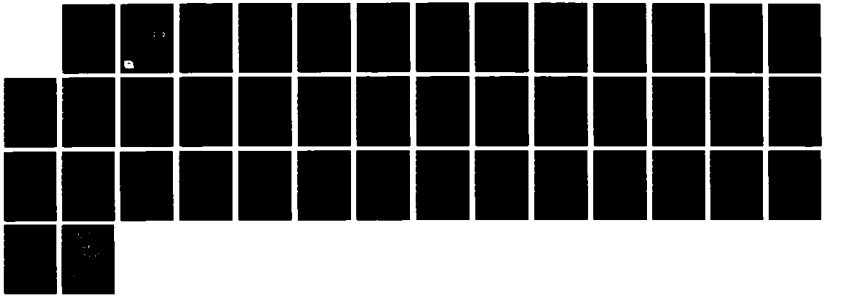
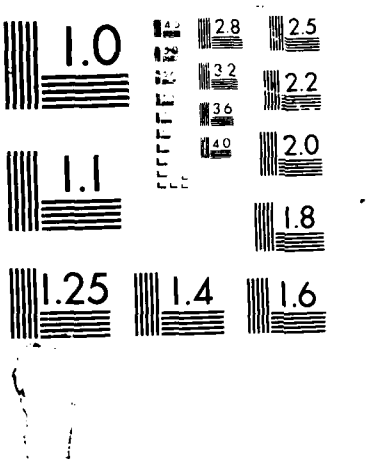
AD-A186 601 HUMAN ENGINEERING IMPACT ON THE STARS SEE ARCHITECTURE 1/1
SEE-ARCH-000-0010(U) INSTITUTE FOR DEFENSE ANALYSES
ALEXANDRIA VA E K BAILEY APR 85 IDA-P-1810
UNCLASSIFIED MDA903-84-C-0031 F/G 23/2 NL




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8

IDA PAPER P-1818

HUMAN ENGINEERING IMPACT ON THE STARS SEE ARCHITECTURE SEE-ARCH-008-001.0

Elizabeth K. Bailey

April 1985

Prepared for

Office of the Under Secretary of Defense for Research and Engineering

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, VA 22311

IDA Log No. HQ 85-29654

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Office of the Under Secretary of Defense for Research and Engineering. The publication of this IDA document does not indicate endorsement by the client, nor should the contents be construed as reflecting the official position of that agency.

This paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and sound analytical methodology and that the conclusions stem from the methodology.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Public release/distribution unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) P-1818	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION STARS Joint Program Office 8b OFFICE SYMBOL (if applicable) SJPO	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-D5-429 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) Human Engineering Impact on the STARS SEE Architecture SEE-ARCH-008-001.0 (U)	
12 PERSONAL AUTHOR(S) Elizabeth K. Bailey	
13a TYPE OF REPORT Final 13b TIME COVERED FROM _____ TO _____ 14 DATE OF REPORT (Year, Month, Day) 1985 April 15 PAGE COUNT 46	
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Software Engineering Environment; Human Engineering; Human Factors; Software User Interface; Environment Architecture; Joint Service Software Engineering Environment (JSSEE); Software Technology for Adaptable, Reliable Systems (STARS)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this study is to address human engineering issues as they relate to the Joint Service Software Engineering Environment (JSSEE) and, in particular, to discuss their impact on the development methodology, supporting tools, and the JSSEE architecture. The purpose is not to address what constitutes "good" human engineering of an environment except at a very high level, since much much of that can only be answered within the context of the entire environment. Rather, the objective is to outline the approach that is needed to ensure that human engineering goals are incorporated in a meaningful way and that human engineering methods and tools are successfully integrated into the overall development methodology.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ■ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

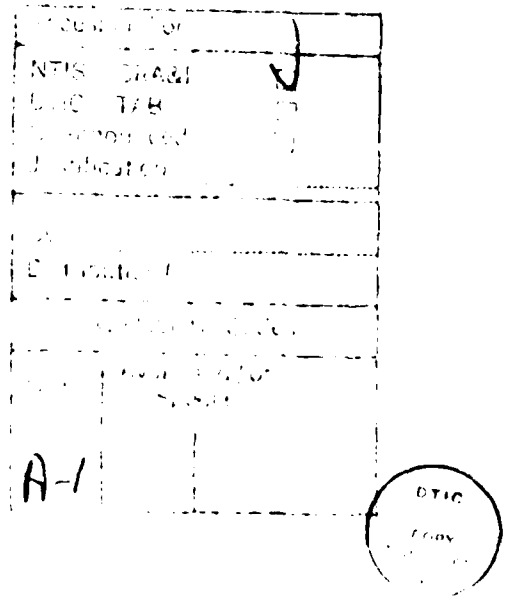
SECURITY CLASSIFICATION OF THIS PAGE

IDA PAPER P-1818

HUMAN ENGINEERING IMPACT ON THE STARS SEE ARCHITECTURE SEE-ARCH-008-001.0

Elizabeth K. Bailey

April 1985



IDA

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-D5-429

Table of Contents

PREFACE

The purpose of this study is to address human engineering issues as they relate to the Joint Service Software Engineering Environment (JSSEE) and, in particular, to discuss their impact on the development methodology, supporting tools and the JSSEE architecture. The purpose is NOT to address what constitutes "good" human engineering of an environment, except at a very high level since much of that can only be answered within the context of the entire environment. Rather, the objective is to outline the approach that is needed to insure that human engineering goals are incorporated in a meaningful way and that human engineering methods and tools are successfully integrated into the overall development methodology.

Section 1.0 contains an overview of the contributions and responsibilities of the human engineer, especially as these differ from those of the hardware and software engineer.

Section 2.0 contains a discussion of human engineering issues across the life cycle as well as a discussion of various methods which have been developed to address these issues and supporting tools needed.

Section 3.0 contains a lengthy discussion of various evaluation methods, ranging from paper and pencil analyses of design features to formal experimentation with representative users.

Section 4.0 examines a number of concepts commonly associated with human engineering. Requirements that a system be "easy to learn for novice users" or "powerful for experienced users" are vague and unverifiable when stated in that form. It is shown how these requirements can be expressed in the form of precise, testable specifications that can serve in guiding the design, in comparing alternative designs, and as the basis for system acceptance tests.

Section 5.0 contains a discussion of architectural issues. In order for a system to be properly human engineered, it must allow for cycles of design, evaluation, and re-design. This imposes a major requirement on the system architecture which must be such that it allows each specialist (hardware, software, and human factors) to pursue his or her special concerns in a way that is cooperative and non-interfering. In the same way that the software designer has traditionally isolated the software from the effects of various hardware changes, the computational portion of the software should be isolated from the effects of changes in the user interface.

1.0 INTRODUCTION

No matter how powerful or potentially useful the JSSEE may be in terms of its functionality, if it is not usable, its impact will be greatly diminished. The JSSEE represents a special challenge because it will support such a diverse user community and set of activities. Designing a system which is highly usable for all these various users will not be a trivial undertaking.

What does human engineering contribute that is not already contributed by software engineering? Surely there are many software designers who are concerned with building usable as well as useful systems. One contribution stems from a view of the user as an integral part of the system, situated not on the periphery as an "end user" but at the center. The resulting design is viewed and analyzed from the perspective of the user. This perspective represents an important step toward achieving consistency in the design, especially at a global level. The consequence should be a user interface that is designed and evaluated as a coherent whole in all its aspects (command language, error handling, system feedback, help facilties, tutorial aids, etc.) rather than being scattered among different software components and "falling out" as a by-product of the programming task.

A second unique contribution of human engineering is a reliance on user behavior and performance as a major source of input to the design process. The human engineer relies on empirical observation - on observation of what people do, how they behave, what errors they make, how they go about accomplishing a given task, how long it takes to accomplish, and so on - in guiding and evaluating design decisions. The idea of evaluating a system in terms of its effects on user performance is relatively new for computer scientists and is one with which behavioral scientists are more likely to feel comfortable. Moran (1981) expressed the difference in emphasis as follows:

> Computer science, of course, has not been totally oblivious of the computer user. Substantial sub-fields of computer science, such as programming languages and computer graphics, are directly concerned with users and how they communicate and interact with computer systems. The treatment of user behavior, however, is typically cavalier.

It is worth noting that in measuring and evaluating the users' performance, we are in reality evaluating the performance of the JSSEE as a whole since that performance will reflect the combined result of the hardware-software-human system working to accomplish a given task. Thus, the evaluation of user

performance will lead directly to meaningful measures of system productivity.

1.1 Responsibilities of the Human Engineer

Broadly speaking, the human engineer is responsible for the design, evaluation, and maintenance of user interfaces to the system. This may not be a single person but in a system the size of the JSSEE is likely to be an entire team.

To describe the JSSEE as a hardware-software system is incomplete at best. It is a HUMAN-computer system - a fact that should have major consequences for the way in which it is developed. In order to optimize the performance of the system as a whole, one needs to be concerned not only with the performance of the hardware and software components but also with the performance of the human components. It is the job of the hardware and software engineer to evaluate the load on the hardware and software in alternative configurations. By the same token, it is the job of the human factors engineer to evaluate the load (physical and cognitive) on the users and to be concerned with their performance (e.g., learning time, errors). Each specialist relies on specific methods and tools for performing these kinds of analyses. Much of this paper deals with the methods and tools needed by the human factors engineer.

There are four major components involved in human engineering for the JSSEE:

- The first involves the establishment of goals for the user interface. Because different groups of users (and even different activities) have different needs, these requirements must be expressed with respect to specific user groups and activities. An example might be the following: "Ease of learning and ease of retention are of primary importance for project managers using the JSSEE for resource estimation." Note that this requirement applies not only to automated functions provided by the JSSEE but to the various non-automated methods and procedures which will be part of the JSSEE as well. These goals can be translated into specific, verifiable requirements specifications which can serve as the basis for acceptance tests of system usability. It is recommended that this be done at least for those JSSEE functions that support highly critical user activities.

- The second major component is the use of various methods and aids during the design phase to maximize the likelihood of designing a system which meets its usability goals.

- The third major component involves a reliance on evaluation - especially on testing with representative users - to guide the design process. The earlier this evaluation can begin, the less disruptive will be the inevitable design changes. (Technically, this third piece is subsumed by the second but it is discussed separately because empirical evaluation plays such a central role in human engineering work.)

- The fourth major component is the design of a system architecture which isolates the impact of changes to the user interface from other software components. This is particularly important for allowing cycles of design, evaluation and refinement.

The following sections outline issues arising across the life cycle as well as methods which are available and supporting tools needed. The human engineering of human-computer systems is a new field, borrowing concepts from traditional, hardware-oriented human factors engineering as well as from psychology, linguistics, computer science, and other fields. The two sub-fields of cognitive psychology and psycho-linguistics are especially valuable sources of ideas because much of the "load" on the user is more mental than physical.

It is worth noting at the outset that much of the strictly physical and perceptual aspects of workstation design are reasonably well understood (such as the minimum character height required for legibility at various viewing distances). Concerning much of these, a set of guidelines and standards should be straightforward to put together for the JSSEE. However, significant physical issues exist if it is desired to explore substantial variations from current practices (such as wall sized viewing surfaces). And, of course, empirical evaluation is also essential for physical and perceptual aspects. Nevertheless, the design of the more cognitive, conceptual aspects are less straightforward and are not nearly as well understood and that is where the bulk of the human engineering effort must be directed.

It should also be noted that there is no single agreed upon methodology for the human engineering of human-computer systems. Most of the methods discussed in this paper are relatively untried or have been used primarily in research contexts. There is, in addition, an almost total absence of tools to support these methods. This should not be interpreted as reflecting on the value of these methods but, rather, as reflecting on the ad hoc, unstructured way in which user interfaces are typically developed.

The following attempts to point to the pieces of a human engineering methodology for the JSSEE. While it is intended as a

useful starting point, it leaves a significant amount of hard work and creativity to the JSSEE developers. The JSSEE represents an important opportunity to incorporate human engineering in a meaningful way from the earliest stages of development.

2.0 HUMAN ENGINEERING ACROSS THE LIFE CYCLE

2.1 Pre-Software Development and Software Requirements Analysis

The activities which lead to a useful requirements document from a human engineering perspective are much the same as those which lead to a useful document from a hardware or software perspective. The end result should be a thorough understanding of required system functions (manual as well as automated), performance requirements, operating environments, constraints, etc. The earliest decisions concern which system functions to automate and the nature and extent of that automation. These decisions - which require input from hardware, software and human engineering - should be based on a number of factors including costs (both development and operational), reliability, feasibility, and so on. It is the responsibility of the human engineer to understand how the automation of specific functions will impact the way in which users carry out their tasks and, in particular, how they will impact the physical and mental load on various groups of users. This is likely to be difficult in the case of the JSSEE given the wide variation in the operational procedures of different software organizations.

There are additional questions that belong more exclusively to the domain of human engineering and that require an understanding of the needs of specific groups of users and the demands of different types of activities. User interface characteristics that may be desirable for one class of users or for one type of task may not be desirable for another. Ease of learning, for example, may be of overriding importance for inexperienced or infrequent computer users while efficiency and power may be of far greater importance for experienced or every-day users. For tasks that are repetitive in nature, requiring a low level of problem solving (such as routine data entry and text editing), efficiency of physical action (as measured, for example, by the number of keystrokes) may be a major concern. In contrast, for tasks that inherently require a high-level of mental effort (such as some aspects of software design), the concern will be to minimize the user's cognitive load and reduce user errors, in effect, maximizing mental rather than physical efficiency.

An early step in the human engineering of the JSSEE will be to establish design goals for the user interface (or user interfaces). Specifically, this will involve characterizing various groups of JSSEE users and their activities in terms which have consequences for design (e.g., infrequent users, repetitive activity) and, of course, being explicit about the nature of these consequences (e.g., ease of learning, efficiency

of physical action). This information can be obtained through the use of surveys and interviews as well as through the direct observation of users.

An early step in the human engineering of the JSSEE will be to establish design goals for the user interface (or user interfaces). Specifically, this will involve characterizing various groups of JSSEE users and their activities in terms which have consequences for design (e.g., infrequent users, repetitive activity) and, of course, being explicit about the nature of these consequences (e.g., ease of learning, efficiency of physical action). This information can be obtained through the use of surveys and interviews as well as through the direct observation of users.

Simply establishing a set of usability goals is a postive step because it forces the developers to think in detail about what usability means for different users and different activities. These goals can provide a valuable framework for design activities and can, at least, steer the design effort in an appropriate direction. In the case of critical user activities, one may choose to go one step further and transform these goals into precise, testable requirements specifications that can serve as the basis for generating system acceptance tests of usability. Since the idea of usability testing is closely related to the more general topic of user-interface evaluation, we will return to this topic in Section 4.0 after addressing evaluation issues in Section 3.0.

2.2. Design and Implementation

Design and implementation of the user interface is expected to proceed in a much more iterative fashion than the design and implementation of other parts of the software. Evaluation, especially in the form of user testing, should play a critical role throughout the development process. In the following discussion, a general distinction is made between methods to guide the design and implementation of user interfaces and methods for evaluation. It should be kept in mind that the two classes of methods are used continually and iteratively throughout the development.

From a human engineering perspective, the goal of the design and implementation phases is to develop a system which meets the usability goals outlined in the previous phase. There are a multitude of potentially useful features for constructing user interfaces (e.g., menus, command languages, touch panels, icons, pointing devices, function keys, etc.). The challenge lies not only in selecting specific features but in putting them together

so that the design as a whole is coherent and consistent. As Rubenstein and Hersh succinctly express the problem (1984), "Every system has good features. It's just that few have good designs."

Consistency in the user interface is an important determinant of many aspects of usability (i.e., ease of learning, efficiency of use, error-proneness) since it allows the user to make predictions about the behavior of the system on the basis of limited past experience or on the basis of knowledge of analogous domains. A large part of the human engineer's job is to ensure that the system exhibits a consistent and predictable set of behaviors. While it is easy to say that consistency is a goal, it can be difficult to realize in practice. For one thing, consistency can (and should) occur at many levels. We can talk about consistency in the set of physical actions required to elicit a given set of system responses, consistency in the way that analogous functions are accessed across different software products or tools, consistency in the syntax or semantics underlying a given command language, and consistency at a global level in terms of the conceptual framework needed to understand and interpret all aspects of system behavior (system messages, on-line help, etc.). Because the user interface encompasses so much and because consistency, particularly at a global level, can be difficult to achieve, human engineering cannot be effective as a last minute, cosmetic remedy for a poor system design. To be effective, the human engineer must be an integral part of the design team. One would not think of leaving out hardware or software expertise during the early stages of design. It is equally important that human factors expertise be represented.

The designer can turn to several forms of guidance in trying to develop the best design possible. The following represent major categories of help for the designer of user interfaces. Much of this help takes the form of trying to insure various forms of consistency.

- Design Guidelines and Principles

Design guidelines represent one means of summarizing current knowledge and opinion. Several extensive compilations exist. Smith and Aucella (1983), for example, have compiled approximately 600 guidelines from a variety of sources. An example guideline is the following: "The color blue should only be used for background features in a display and not for critical data." Guidelines can be a valuable source of help for the designer, both in the early stages of design as well as much later. There are, however, several limitations in their use. Guidelines can contradict each other and is not often clear what the priorities among them should be. Relatively few have been experimentally validated. Many guidelines are context dependent

- a guideline found to be useful under one set of conditions may not be under another set. Conversely, guidelines which apply across a variety of contexts are likely to be so vague as to be almost useless. "Write good error messages" is an example of one very general but vague guideline - for any given group of users or tasks, it is not always obvious what a "good" error message is. The real problem is knowing when and how to apply guidelines in a specific context and to do so in a way that results in a coherent whole. As Hartson and Johnson (1983) point out, "This work on principles has led to a need for methodologies and tools to facilitate inclusion of these principles in interface design."

Guidance can also be found in the form of project-specific standards. For example, standards may specify the format of system messages or the form of command abbreviations. The primary intent is to encourage consistency across different software functions which are typically designed and implemented by a number of different people.

- Language Models of the User Interface

Substantial interest has been directed to the study of human-computer interaction from a language viewpoint. The obvious implication of this perspective is that one can represent and analyze that interaction in terms of its lexical units (e.g., button presses, toggle switchings, key presses), its syntax and its semantics (pointing, deleting, etc.). The very attempt to represent the dynamic, two-way dialogue between the user and the computer provides a much needed framework for describing and analyzing the design. The value of a language model is expressed by Hartson and Johnson: "Without a model of interaction, the elements of the interface have no cohesion but are simply a group of random, unconnected messages, displays and user actions. This lack of a framework leads to dialogue development procedures that are unstructured and random as well."

An example of the use of a language description in analyzing a user interface for consistency is provided by Reisner's work (1981). Reisner used a formal language grammar (BNF) to describe the set of user actions required to operate each of two versions of an interactive graphics system. The two versions contained identical functions but differed in their user interface. On the basis of differences in the measured complexity of the grammars for the two versions, Reisner was able to make explicit predictions about differences in ease of learning. She was also able to point to specific cases of inconsistencies in the underlying production rules (conceptually similar commands required conceptually dissimilar sequences of user actions). She predicted that these inconsistencies would lead to a high rate of user errors. Both sets of predictions were confirmed in a

subsequent experiment. By describing the set of user actions as a language, Reisner was able to pinpoint design difficulties on the basis of a pencil-and-paper analysis.

- Conceptual Models

There is ample evidence that the knowledge that users have about any given system is in the form of a "conceptual model" - a model that the user adopts of the system's objects, operations, relationships, and terminology. In trying to learn a new system, the conceptual model typically takes the form of an analogy in which the computer system is viewed as being like something else with which the user is already familiar. The model provides a framework for the user in which to organize and interpret the behavior of the system and to make predictions about how the system will behave in new circumstances. To the extent that these predictions are accurate, the conceptual model is useful. A major objective in designing any system should be to facilitate the development of a useful, consistent conceptual model, either by patterning the behavior of the system after a common analogy or by creating a new model which is consistent in all its aspects (i.e., command names, error messages, help facilities, user documentation).

An example of a conceptual model in action is provided by a study by Mack, Lewis, and Carroll (1983) who observed a group of typists learning to use a word-processor for the first time. The conceptual model consistently adopted by the typists was based on a typewriter, an analogy that served both as a source of help and as a source of interference. Mack et al. provide the following description of the learning behavior of the users in that study:

> It is natural that learners try to relate how the text-processing system works to what they know about typewriters. On the positive side, participants did not have trouble typing basic alphanumeric characters or with the fact that characters appear on a video display rather than on paper. Other functions, however, reveal inappropriate expectations about how the text-processing system operates.
>
> Participants were surprised that the SPACE bar, BACKSPACE and RETURN keys not only moved the typing point but also changed material they had already typed...Participants also tried to relate the screen into which text was entered to a blank sheet of paper in a typewriter, but had trouble doing so. They wondered where the top, bottom, and margins were, since the screen was smaller than a complete page. They wondered how to set the margins, since the familiar mechanical controls of the typewriter were missing.

> There was message information on the screen that they had not typed and they wondered whether this would appear on their finished printed document.

The typewriter analogy also left the typists ill-prepared for the terminology of the text-processing world:

> One participant, for example, wondered what a cursor was after several hours of using the system. Another wondered "who" the printer was. Yet a third wondered if she was the printer.

Lest the reader think that these kinds of difficulties afflict only the first-time computer user, Nielsen (1984) found exactly the same kinds of problems when professional programmers and computer science students were faced with the problem of learning to use a syntax-directed editor with windowing capabilities; all had extensive experience with line-oriented and screen-oriented editors.

> While the learners in the Mack et al. experiment generalized from their knowledge of typewriters, our learners generalized from their knowledge of traditional "glass-TTY" interfaces...
>
> For example, all of our users had problems the first time they created a long line in a program. The line would not fit inside the window and part of the line was therefore invisible. It took some time for our learners to realize that the missing part of the line had not been dropped by the editor but was still present inside the computer and could be worked on. They had trouble thinking of the concept of horizontal scrolling, which of course may also be due to their considering a program a one-dimensional structure that can only be scrolled vertically...
>
> One learner was puzzled that the editor as explained above did not show very long lines in full and proceeded to look through the user's guide for information on how to switch the "word wrap" mode. The editor did not have such a mode and the user was supposed to use scrolling instead. But it took him a long time to realize this.

The explicit attempt to design a useful conceptual model will go a long way toward achieving the goal of global consistency. All aspects of the system's behavior should be consistent with that conceptual model. For example, error messages should be

expressed in terms of the operations and objects of the conceptual model and not in terms of the underlying hardware or software.

It may be the case that more than one conceptual model is desirable for the JSSEE, depending on the tasks and users. This is an issue that can only be decided by further study, perhaps in the form of experimentation with representative users. The starting point will be to understand the backgrounds and terminology of the different classes of users and the conceptual models they are likely to adopt. Armed with this knowledge, one may choose to explicitly reinforce one or more models. As an alternative, one at least has a basis for making predictions about the types of difficulties learners are likely to run into based on inappropriate analogies. One can then design tutorial aids and other forms of user documentation to minimize those difficulties.

2.3 Operational Phase

One of the primary activities of the human engineer during the operational phase is to obtain feedback about system usability. In addition to the use of surveys, interviews, and on-line consultants, the automatic collection of usage statistics can provide an important source of information. These statistics can be used to identify the system functions which are used infrequently. One can then ask whether this reflects a lack of usefulness or a lack of usability. If the problem is one of usability, one may choose to re-design that portion of the user interface.

2.4 Tools for the Human Engineer

The general classes of tools needed by the human engineer include:

- tools for simulating and prototyping user-computer dialogues.

These could, for example, accept as input a BNF or state-transition diagram with associated actions and produce as output a simulation of the dialogue between the user and the computer.

- tools and associated databases for creating, modifying and storing displays of various types (graphics, text, forms, voice,

menu, keypad and touch-panel displays) and general displays definitions to ensure consistency across the system (Hartson and Johnson, 1983).

- tools and associated databases for defining input languages of various types (which can be in the form of menu selection, keypad key, command string, forms filling, voice input). Hartson and his colleagues at Virginia Tech are working on a tool which develops a formal definition on the basis of example productions. Thus, it requires no special training in formal language definition.

- tools for evaluating user interfaces. These are discussed in Section 3.3.

3.0 EVALUATION

Evaluation plays an especially critical role in human engineering work, particularly in the form of testing with actual users. The knowledge does not exist to guarantee an acceptable user interface solely on the basis of design guidelines, standards or anything else. There are too many variables, in terms of design alternatives and user characteristics, whose effects are difficult to predict. It is essential to obtain objective feedback about the design, especially at an early point in the development before change becomes prohibitively expensive.

There are two basic approaches to evaluation. In the analytical approach, the evaluation is based on an analysis of features or properties of the user interface; the empirical approach focuses, not on design features, but on the user's behavior in interacting with the system. Both approaches can vary considerably in terms of their formality and objectivity. Neither is the single best approach for evaluating the JSSEE - both can be useful (and should be used) for particular purposes and at particular points in the development.

3.1 Analytical Evaluation Methods

Analytical methods represent an "armchair" approach to evaluating user interfaces. An example of a somewhat subjective evaluation would be a check for adherence of a user interface to design guidelines. An example of a more rigorous analytical evaluation is represented by Reisner's (1981) work, described in Section 2.0, analyzing the grammar underlying two different user interfaces to the same functional system. Lindquist (1985) has proposed a technique based on the application of conventional software metrics for evaluating the complexity of the control structure underlying human-computer dialogues. He assumes that measured complexity is related to various aspects of user performance such as ease of learning although no attempt has been made to validate this relationship.

One of the strengths of the analytical approach is that it generates specific predictions about hypothesized sources of difficulty in a design and, thus, can serve a valuable diagnostic role (for example, it not only predicts that a given user interface is difficult to learn but is explicit in pointing to the source of the difficulty). In addition, substantially less time and money are required to evaluate a user interface based on an analysis of its properties or features as compared to even the most rudimentary evaluation with real users. The main problem with the approach is that it generates hypotheses - not observations - about user performance. These hypotheses still

require validation with actual users. Given such validation, one has some degree of confidence in the validity of the analysis for similar systems in the future.

3.2 Empirical Evaluation Methods

The analytical approach represents a valuable source of predictions concerning user behavior but the only truly objective and sure way to discover whether or not a design is usable is to try it out with representative users. If, for example, one is interested in evaluating a given user interface for ease of learning, one cannot do this solely on the basis of an analysis of design features or any other attributes of the user interface. There is no single feature or combination of features that will guarantee ease of learning. Instead, one must observe users actually learning to carry out a task with the system. As with analytical methods, empirical methods differ in terms of their formality and rigor, ranging from informal, observational studies to formal, standardized tests. The primary criterion for deciding on the degree of formality necessary is whether the purpose is to evaluate a single design or to choose among a set of alternatives (Table 1). If the purpose is to evaluate a single design, an observational study is sufficient. If the purpose is to choose among design alternatives, one must turn to a more formal set of procedures. Both of these methods are discussed below. First, however, is a brief discussion of the minimal requirements for any empirical evaluation, regardless of the degree of formality.

	EVALUATION METHODS		
	NON-EMPIRICAL	EMPIRICAL	
PURPOSE	Analytical	Observational	Standardized
Focus on a single design	X	X	
Compare Alternative Designs	x		x

Table 1: Evaluation Alternatives and their Use

3.2.1 Minimal Requirements: Interfaces, Users, and Tasks

In order to conduct an evaluation, one obviously needs something to evaluate. Fortunately for the human engineer, one does not need to wait for a completed system but can evaluate a prototype or simulation at an early point in the development. It is not even necessary to have a computer in order to simulate key aspects of a user interface. Rubenstein and Hersh (1984), for example, suggest having pairs of users work together to simulate various aspects of human-computer dialogue, with one person taking the role of the user while the other plays the part of the computer. This informal procedure requires minimal resources and yet can be extremely helpful in pointing to sources of confusion.

Simulation can also be used in the context of a much more rigorous evaluation. Gould, Conti, and Hovanyecz (1982), for example, conducted a controlled experiment using a simulated "listening typewriter". This typewriter allowed users to dictate a memo, letter, or other material rather than typing on a keyboard. For the simulation, users spoke into a microphone; they were seated in front of a CRT which displayed the words as they spoke. Intervening between the microphone and the CRT (but hidden from view of the user) was a typist who functioned as a sophisticated speech-recognition system. With this arrangement, Gould et al were able to study the effects of various parameters of the system on user performance. The parameters studied included vocabulary size (1000 words, 5000 words, or unlimited) and whether the typewriter was capable of recognizing continuous speech or only isolated words (which required the users to pause after each word). The performance measures of interest included the judged quality of the resulting letter or memo. This method of simulating user interfaces, sometimes referred to as the "Wizard of Oz" technique, allows one to study the effects of variations in systems that don't yet exist. For this reason, it can also be a useful technique for evaluating different interface concepts during the very early stages of conceptualizing a system, prior to any attempt at formal requirements analysis.

No matter how informal the evaluation, it is important that the users be representative and that they have no vested interest in the outcome of the evaluation. The designers of a system should never play the part of the users during an evaluation, not only because of their unavoidable interest in the outcome but also because they know too much about the system. Nielsen (1984) makes this point in discussing the results of a study he undertook looking at programmers learning to use a syntax-directed editor for the first time:

> Since the intended users of such an editor are programmers and since the editor designers are themselves programmers, the designers may feel that they are justified in doing the human factors evaluation of their editor using themselves as subjects...The editor developers and the editor users are two different kinds of programmers however. In the course of designing and implementing their editor, the developers form a detailed actual conceptual model of the editing principles implied by a structure editor as opposed to a text editor. The users do not have this model, and the present experiment showed that ordinary programmers may indeed have several problems when faced with a totally new way of editing.

No matter how informal the evaluation, it is important that the tasks that are carried out during the evaluation be representative. If they are not representative, the evaluation will not yield useful, valid information about the usability of the system being built no matter how carefully constructed it is in other respects. Identifying representative tasks is not a trivial matter but will be much more likely to succeed if a thorough job of requirements analysis has been carried out. Moran (1981) points out the difficulty but also the value inherent in attempting to identify a set of representative tasks: "...there is at present no methodology for making this choice; it must be done informally. For this reason, perhaps, computer systems usually seem to be designed with no clear definition of the tasks for which they are to be used. Precise specification of the population of tasks that a system is to address is a big step toward accounting for the user in system design."

3.2.2 Informal Empirical Evaluations

As noted above, when the purpose of an evaluation is to focus on a single design rather than to compare design alternatives, one does not need to resort to a set of formal procedures. Instead, one can learn a great deal from relatively informal observations of users interacting with the (real or simulated) system. Videotaping can be a useful tool for this form of evaluation. Rubsenstein and Hersh suggest that "Videotaping is much less prone than manual note-taking to errors of omission or to biases in recording. One tape of someone sitting back, looking confused, and saying 'I have no idea of what to do,' is worth a dozen technical papers on the analysis. Conversely, user success is very convincing on tape."

For tasks that have a large component of mental rather than physical activity, there are techniques which attempt to gain accessibility to the users' thought processes so that one can better identify stumbling blocks to effective user performance. One such technique involves asking users to verbalize out loud their thoughts and problems. This technique is also useful in identifying the user's conceptual model. Another technique (suggested by Rubenstein and Hersh) is to have pairs of users working together to carry out a given task -the evaluator can learn a great deal about their confusions and problems by observing their efforts to explain things to each other.

The major advantage of an observational evaluation is that a great deal can be learned without committing substantial resources. In addition, it does not require any specialized training in human performance measurement on the part of the evaluators. The primary disadvantage is that this form of evaluation does not allow us to compare alternative designs in any way that is rigorous and repeatable. For that, we must turn to a standardized evaluation.

3.2.3 Formal Evaluations

When the purpose of the evaluation is to compare alternative designs, then the goal is not only to ensure that the evaluation is objective and meaningful but also that the results obtained for one design can be directly compared with the results obtained for an alternative. A major requirement for making this comparison is consistency in the way that the evaluations are conducted - the procedures for the evaluation must be carried out in a standard way for each design that is evaluated. A second requirement is to control any and all factors which could influence the outcome so that any differences observed between alternatives can be attributed to the designs themselves and not to other extraneous factors. Both of these requirements are achieved by adhering to the methodology of controlled experimentation.

A large part of the challenge in attempting to compare alternative designs in terms of a seemingly fuzzy attribute like "usability" is to be sufficiently precise in defining exactly what is being evaluated so that the comparison could be repeated by a totally independent person with the same outcome. To achieve this precision, it is necessary to express the definition in terms that are observable, measurable and, hence, repeatable. For this we must turn to an operational definition of usability. The following discussion of operational definitions is largely taken from an earlier paper written for the APSE Evaluation and Validation Team (Bailey and Kramer, 1984).

The concept of an operational definition was developed by experimental psychologists who were studying the effects of various factors that intuitively seem important but were difficult to pin down (such as "motivation" or hunger"). What does it mean to say that an animal in an experiment is "highly motivated" or "very hungry"? Even if everyone has an intuitive sense of the meaning of these terms, intuition does not provide the precision needed to allow an experimenter to replicate the conditions of the experiment or to compare any two sets of results. Instead, it was realized that conditions such as "hungry" or "not hungry" must be defined by the experimental operations that were used to produce them (for example, 24 hours of food deprivation for "hungry" versus one hour of food deprivation for "not hungry"). This is akin to asking what it means to say that a particular user interface is "easy to learn" to "easy to use". Intuitively, we may have a strong feeling for what these terms mean but our feelings do not provide an adequate foundation on which to conduct a comparative evaluation. Instead, we must give ease of learning an operational definition along the lines of the following: "The ease of learning System X will be defined by the average score obtained by a sample of 20 users with the following characteristics on the following set of tasks under the following conditions..."

Shackel (1981) gives an excellent example of an operational definition from a domain outside of computer science. The definition makes very precise what is meant by saying that a medicine container is "childproof". In reading the definition that follows, note that it is TESTABLE because it is expressed in operational terms. We will return to this example in Section 4.0 when we discuss the generation of testable usability requirements for the JSSEE.

> ...consider the U.S. legislation (Federal Register, 1971) and the British Standard (1975) on childproof medicine containers, both of which are essentially concerned with (human) performance...issues; they specify that at least 85% of a test panel of children shall be unable to open the containers before a demonstration, and at least 90% of a panel of adults shall be able to open and properly reclose them following written instructions only.

While childproof is defined as a pass-fail criterion in this example, one could easily compare two or more containers in terms of the degree to which they are childproof by following the procedures specified as part of the operational definition.

In order to compare two or more designs in terms of usability, what is needed is an analogous approach which specifies the population of interest, the tasks, the conditions

of observation, and the measures used for the evaluation. In short, all components of the evaluation must be specified in sufficient detail so that an outside person could repeat the evaluation and expect to obtain the same results (within the bounds of statistical sampling error). This specification must include at least the following:

- characteristics of the user population to be sampled
- minimum sample size
- procedures for selecting the sample of users
- set of tasks to be used ("benchmarks")
- conditions of instructions, including the availability of user documentation
- measures to be collected (e.g., time, errors, number of tasks successfully completed, user-provided ratings)
- procedures for their collection
- any other information required to allow others to repeat the evaluation as closely as possible

An operational definition transforms a fuzzy concept into a precise one. One can, of course, argue with any given definition on the grounds that the users or tasks are not representative, that the measures are not appropriate, and so on, but at least one has something concrete and specific to argue.

It is important to recognize that each part of an operational definition is important. For example, consider the characteristics of the user population to be sampled - one might find different patterns of results depending on the population of users represented in the sample. In such a case, one could not simply state that one design is superior to another. Rather, one can only claim superiority within the bounds of the conditions that were represented in the evaluation. This has obvious consequences for the generalizability of evaluation results. If one is interested in whether a given set of results applies to a different population of users or a different type of task, the evaluation must be repeated using that population of users or that type of task.

The advantage of a standardized evaluation is that it allows one to compare alternative designs in an objective, rigorous fashion. The disadvantage is the expense and the extensive preparation time involved. Specialized training in human performance measurement is required to plan and carry out the

evaluation since various forms of bias may go unnoticed by an evaluator untrained in experimental methods. A thorough discussion of standardized evaluations of usability can be found in Bailey and Kramer (1984).

3.3 Evaluation Facility Required for the JSSEE

The particular questions and comparisons of interest will change at various points in the JSSEE development. The early questions are likely to center around the appropriateness of one or more conceptual models. Later issues will involve more specific aspects such as the selection of command names or icons, the design of specific displays, and so on. As an example of the kinds of issues which can be addressed by empirical evaluation, Table 2 (adapted from Bewley, Roberts, Schroit, and Verplan, 1983), contains a partial list of the design issues that were the subject of formal experimentation in the development of the Xerox 'Star' Office Workstation. A total of 15 experiments were conducted, using more than 200 users and requiring more than 400 hours of test time, by an evaluation group which averaged six people over a three-year period.

TEST TOPIC	NO. USERS	TOTAL HOURS	IMPACT
Keyboard (6 layouts)	20	40	Led to design of keyboard
Display	20	10	Specified display phosphor and refresh rate
Tab-indent	16	16	Caused redesign of tab and indent functionality
Labels	12	6	Caused change in property sheet and keyboard labels
Property Sheets	20	40	Identified potential interface problems and redesigns
Fonts	8	6	Led to decision on screen-paper coordination

Icons	20	30	Led to design of icons
Initial Dialogue	12	36	Led to design of training facilities and materials
HELP	2	6	Validated HELP design ideas
Graphics	10	65	Led to redesign; validated new design
J-Star Labels	25	25	Led to design of keyboard labels for Japanese Star

Table 2: Partial Listing of Empirical Evaluations for the Xerox 'Star' Office Workstation

The evaluators involved in the development of the Star (Bewley et al.) provide a testimony to the value of empirical evaluation, in conjunction with the use of design principles and analytical evaluation:

> The impact...has been a pervasive set of small and large changes to the user interface. The amount of difference these changes made is, of course, impossible to assess, but the quality of Star's user interface is well known. It has won an award as the 'friendliest' computer system of 1982, as judged by Computing magazine. Imitators...are starting to have a major impact on the marketplace. We can only take this as a ratification of Star's design process, a rich blend of user interface principles,... analysis, and human interface testing.

The evaluation facility for the JSSEE will require, at a minimum, the following capabilities:

- tools for constructing prototypes and simulations of user interfaces

- videotaping equipment

- facilities for recording user interaction. These can vary from logging facilities to capture every keystroke to facilities to capture much more general kinds of information such as a summary of the different commands used or the time required to successfully complete a given task. In general, the data of

interest and, hence, the logging facilities required will depend on the question being addressed for any given evaluation. In the course of designing a text editor, one might be interested in evaluating keystroke efficiency and capturing data at that level. One may also be interested in recording and time-stamping all input so that the entire sequence may be replayed and examined in detail for sources of user problems although, as a general rule, this yields massive amounts of data which can quickly become overwhelming.

- facilities to ensure consistency in the procedures (e.g., a standard way of presenting instructions to the users). This is a requirement only for conducting formal, standardized evaluations.

4.0 GENERATING USABILITY SPECIFICATIONS

Section 2.0 contained a discussion of the value of identifying goals for the design of user interfaces to the JSSEE. It was also pointed out that one can go beyond the establishment of high-level goals (such as ease of learning, efficiency of user action) to the generation of specific, testable specifications for usability. The recommendation was made that this be done for a subset of JSSEE functions - specifically, for those that support highly frequent or critical user activities. These specifications can serve not only as the basis for generating system acceptance tests, but equally important, they can serve as explicit criteria for comparing design alternatives much earlier in the development. Operational definitions of usability provide the foundation for these specifications because it is the operational nature of these definitions which make them sufficiently precise so as to be testable. The only addition necessary will be threshold values for each operational definition.

Consider the example of the childproof medicine container which was presented in Section 3. This definition, which actually defined childproof as a particular combination of threshold values, serves as a precise, testable specification of exactly what it means for a container to be childproof. One can take the same approach in the specification of usability, an idea previously suggested by Shneiderman (1983) and by Kruesi (1983). For example, beginning with ease of learning as a goal for a text editor, one can express this in the form of an operational definition:

> X% of a sample of N users with the following characteristics must successfully learn to carry out the following basic editing functions (e.g., opening a new file, inserting text, deleting text, saving the file, printing the document) within a period of two hours using only the vendor-supplied tutorial material for instructions.

The specific aspects of user performance that can be specified include: the time to complete a task, the number of errors made, and the number of functions successfully completed in a given time period. In addition, one can measure (and therefore specify in advance) various aspects of the user's subjective reactions to a system. There are times when these subjective reactions are more important than the observable aspects of a user's performance. For example, judged ease-of-use is more important for discretionary users (users who have a choice between using the JSSEE and resorting to manual methods or another automated system). No matter how impressive the performance in terms of time or errors, if the discretionary user

finds the system more difficult to use than the alternative, it simply will not be used. (This provides another example of why a good job of requirements analysis is important. One of the relevant dimensions along which to characterize users is the degree of discretion.)

Clearly, the larger or the less understood the task, the more difficult it will be to write meaningful specifications of user performance. But it is, at least, worth a try for a subset of the activities to be supported by the JSSEE. In the course of attempting to write these specifications, one will be required to think long and hard about the users and the activities that the JSSEE is intended to support. For cases in which it is not possible to specify acceptable levels of user performance (which will probably be many), one might run a series of pilot studies with existing systems to gain a feeling for acceptable performance levels. In the course of conducting these studies, one can also learn a great deal about design characteristics which appear to contribute to or interfere with various levels of user performance.

5.0 ARCHITECTURAL ISSUES

One of the main themes of this paper is the need for early and continuous evaluation of user interfaces. In order to incorporate the inevitable changes that are suggested by these evaluations, one needs an interface that can be changed rapidly, easily and without adversely affecting the rest of the system. Hartson and his colleagues at Virginia Tech have argued that a major stumbling block in the attempt to design a flexible user interface is the intermingling of the parts of the software that communicate with users (the dialogue component) with the parts that constitute the basic functionality (the computational component). Yunten and Hartson (1983) present a real-life example of the consequences of this intermingling:

> A major vendor recently marketed an office automation system which prides itself on 'ease of use' of its interface. Yet this statement is used to guide the user in accessing the help facility:
>
> 'The comma key on the minikeypad is the HELP key for forms. While in the ABC-Style Editor and Calendar Management, use PF2 for HELP; use "H" for HELP while in the Desk Calculator; use the "GOLD" key plus an "H" key while using the XYZ-Style Editor. ...By the way, if you need help creating a document, it is better to be in the Word Processing Menu when you press HELP rather than in the main menu... It is a good idea to remember the location and purpose of each key mentioned above.'
>
> Of course, the user must already know how to receive HELP in order to get the above message. The explanation given for this inconsistency was that, because the dialogue was so entangled with the computational code, it was unbelievably difficult to change something as seemingly simple as which key to use for HELP.

Hartson has argued that, by separating the dialogue component from the computational component, one creates an architecture which allows these two components to be developed independently and in parallel. This separation gives explicit recognition to the fact that there are really two distinct sets of problems to be addressed in the design of any interactive system. One set of problems belongs to the domain of computer science and has to do with the correct and efficient functioning of the computational portion of the software. The other set belongs to the domain of human engineering and has to do with effective communication between the user and the software.

From the perspective of the computational part of the system, when user input is needed to carry out a given function, the requirement is really to obtain a token value - the fact that the token must be obtained from the user is irrelevant to the problem at hand. With the proposed architecture, all interaction with the user - including prompts, displays of all types, input validation, error messages, on-line help and tutorials - is contained within the dialogue component. It is the concern of the dialogue component to communicate that request to the user, to obtain and validate the specific input and to then pass on the token value to the computational component.

Hartson has argued for a parallel separation in the personnel skills required to design, implement, and test these two components. The dialogue component falls directly into the camp of the human engineer (whose role is that of a "dialogue author") while the computational component is the responsibility of the programmer. This division of responsibilities relieves the application programmer from being side-tracked with validating user input, writing appropriate error messages and other forms of feedback. It also encourages the development of the user interface as a unified whole by a specific person or group of persons with human factors and communications skills. Hartson and his colleagues have proposed an entire development methodology, complete with supporting tools sets, around the separation of the dialogue from the computational component. The objective of this methodology is to provide a common framework and notation for human engineering and software engineering and to facilitate the efficient and smooth integration of these two disciplines.

The interaction between the user and the system can be highly variable with possibilities for all kinds of incorrect input. The dialogue component must include facilities to handle this variation, such as error reporting and help facilities. In contrast to the highly varying nature of the interaction between the user and the dialogue component, the interaction between the dialogue and computational components is relatively fixed and can be formally specified. The ability to formally specify the interface is what allows for dialogue independence. Once the interface (control flow and data flow) between the two components has been established, the human engineer is free to experiment with all kinds of alternatives in the dialogue component without impacting the rest of the system so long as their common interface does not change. Not only are the dialogue and computational components logically and physically separate at design time, further separation is attained during run time by their concurrent execution as autonomous processes. Thus, changes in one component do not require a change or a re-linking of the other component. In multi-processor implementations, this may add to the efficiency of execution as well.

The Hartson work is of interest because it represents the most comprehensive effort to incorporate human engineering as part of a larger system development methodology. The ideas proposed deserve closer study for the JSSEE. There are two problems of particular importance to the JSSEE that can be addressed by this approach. One is the problem of device dependencies which is likely to be a major issue given existing hardware which ranges from teletypes to screen-oriented character displays to high-resolution graphical displays. Dialogue independence allows for drastic variation in user interfaces for the same set of functions. Thus, interfaces can be developed for a wide range of hardware and interaction styles.

A second problem that must be addressed involves the addition of application-specific and service-specific tools to the JSSEE. There is potentially a very real problem that these additional tools will lead to serious inconsistencies in the user interface, at a number of levels. One possible solution is to require any and all add-on tools to contain only the computational portion and to conform to a formal specification for the interface between the computational and dialogue components. It will then be a human engineers' job (residing perhaps at the Software Engineering Institute) to write the dialogue for that added tool.

6.0 RECOMMENDATIONS

1. Include human engineering as a major emphasis in the development of the JSSEE.

A basic guiding principle should be a recognition of the user as an integral part of the human-software-hardware system. For the entire system to function effectively, the design must be represented and evaluated, not only from the perspective of the hardware and software, but also from the perspective of the user. The behavior of the system should be consistent at all levels and in all its various aspects. Several general classes of design methods and aids were discussed, all of which should be used in the development of the JSSEE, particularly in helping to insure consistency. They include:

-- language models of human-computer interaction
-- user conceptual models
-- design guidelines

In addition, empirical evaluation, including both informal observational and formal studies, should provide a major source of input to the design process. The evaluation of various design alternatives, in terms of their impact on user performance, should begin early and should continue throughout the development and evolution of the system. In order to evaluate user performance in a way that is meaningful and objective, an evaluation facility will be required (which is discussed as Recommendation 5 below).

2. Conduct an empirical analysis of JSSEE users and their tasks.

The output of this activity should identify the following:

- high priority user tasks

- user and task characteristics

- design goals and constraints

The JSSEE Operational Concept Document (OCD) contains a high-level description of various groups of JSSEE users and their responsibilities and tasks. A major next step is to validate these descriptions and to add considerably more detail about the tasks and users in terms of characteristics which have implications for the design of user interfaces to the JSSEE. These characteristics can then be used in generating usability goals (e.g., ease of learning) for specific combinations of users and tasks as well as in identifying design constraints (e.g., minimal typing). These can also help in gaining insight into the conceptual models likely to be adopted by various groups of

users. The establishment of design goals will force the developers to think in detail about what usability means for different users and different activities. These goals can provide a valuable framework for design activities and can steer the design effort in an appropriate direction. In the case of critical user activities, these goals should be transformed into precise, testable requirements specifications that can serve as a basis for generating system acceptance tests of usability.

An additional output of this analysis should include the identification of high priority user tasks. These priorities should be determined on the basis of task criticality and frequency. The set of high-priority user tasks will provide a focus for the human engineering effort; these tasks should provide the basis for generating a specific set of benchmark tasks for comparing design alternatives in terms of their effects on user performance and for comparing the JSSEE to other systems.

This empirical analysis will most likely be accomplished through the use of surveys, interviews and direct observation of user activities. The recommended first step is a study outlining a feasible approach for gathering, analyzing, and summarizing this information within reasonable resources limitations.

3. Express design goals in the form of operational definitions.

Through the use of operational definitions, seemingly fuzzy concepts such as "ease of learning" or "ease of use" will become precise and testable. This will be essential for comparing design alternatives in terms of their "usability". These operational definitions will also form the foundation for system acceptance tests of usability.

4. Investigate in detail the architectural implications of dialogue independence.

A recommended early activity is to take a closer look at the work being done by Hartson and his colleagues at Virginia Tech. Their work represents a comprehensive, well-thought out approach to the successful integration of human engineering requirements and methods with more traditional software engineering concerns. It is worth looking not only at the concepts surrounding the separation of the dialogue and computational components but also at the corresponding tool sets for the human engineer and programmer.

The requirements and potential specifications for the interface between the dialogue and computational portions of JSSEE should be immediately explored to establish feasibility. Present

efforts should be reviewed (e.g., Apple (Bralsicke, 1985), Imperial Software Limited, and IBM (Carlson, Rhyne and Wellen, 1983). The earlier such an interface can be established the better. In any case, a high level of confidence is needed in such a specficiation before final implementation of JSSEE begins. The implications of the need for extensibility in the dialogue portion of the JSSEE should be an important part of this investigation.

5. Establish a JSSEE Human Engineering Evaluation Facility.

In comparing alternative designs for the JSSEE, an important criterion should be usability, particularly as measured by user performance. In measuring user performance, one is, in reality, measuring the performance of the hardware-software-human system as a whole. Hence, the evaluation of usability will provide a direct measure of system productivity.

The generation of operational definitions of usability will provide the foundation for rigorous, objective comparisons of design alternatives. For both formal, standardized evaluations as well as informal, observational analyses, facilities will be needed for measuring user performance and for recording user interaction; these will include videotaping as well as computer instrumentation to capture information about keystrokes or commands used, their sequence, timing, and so on. Facilities for creating simulations and prototypes of user interfaces will also be needed.

6. Include the human engineer as a JSSEE user in the Operational Concept Document (OCD).

This paper is concerned with the human engineering of the JSSEE. The issues, methods, and tools discussed are not specific to the development of software support environments but apply equally to the development of any system which interfaces to human users. It is equally important that methods and tools be included within the JSSEE to support the human engineering of mission critical computer resource (MCCR) systems. The first step toward that end is to explicitly include the human engineer as a JSSEE user in the OCD. The following paragraphs provides one possible description of the human engineer as a JSSEE user.

The human engineer is responsible for the design, implementation, evaluation, and maintenance of user interfaces to a system. The human engineer analyzes user interfaces with respect to the physical and mental load on the user and with respect to their effect on user performance. The human engineer

also contributes to the design of the operational procedures (management, administrative, and technical) in which the users and tools are embedded.

Human engineering methods can be categorized into four major classes: (1) use of design guidelines, standards and principles, (2) use of language models of the user interface, (3) explicit consideration of the conceptual models likely to be adopted by users, and (4) empirical evaluation (user testing). The general classes of automated support for the human engineer include tools for simulating/prototyping aspects of user interfaces, tools for creating and modifying displays and display templates of all kinds (e.g., text, graphics, forms, menus), tools for generating formal descriptions of command languages and other types of user input, tools for logging and time-stamping user interaction (with a prototype or with an implemented system), and databases for storing relevant information (e.g., displays, display templates, language definitions, data generated from logging user interaction).

REFERENCES

Bailey, E.K. and Kramer, J.F. "A Framework for Evaluating the Usability of Programming Support Environments", Manuscript submitted for publication.

Bewley, W.L., Roberts, T.L., Schroit, D., and Verplank, W.L. "Human Factors Testing in the Design of Xerox's 8010 'Star' Office Workstation". In CHI'83 Conference Proceedings: Human Factors in Computing Systems, ACM, December 1983, pp. 72-77.

Braesicke, J.D., et al., "Working Group Report on User Interfaces", Proceedings of the ACM AdaTEC Future Ada Environments Workshop, to be published in SIGAda Ada Letters and SIGSOFT Software Engineering Notes, Spring, 1985.

Carlson, E.D., Rhyne, J.R., and Wellen, D.L., "Software Structure for Display Management Systems", IEEE Transactions on Software Engineering, Vol. SE-9, No. 4, July 1983, pp. 385-394.

Gould, J.D., Conti, J., and Hovanyecz, T. "Composing Letters with a Simulated Listening Typewriter", in Proceedings of the Human Factors In Computer Systems conference, ACM, Washington, D.C., March 1982, pp. 367-370.

Hartson, H.R. and Johnson, D.H. "Dialogue Management: New Concepts in Human-Computer Interface Development", Manuscript submitted to ACM Computing Surveys (revised version), March 1984.

Kruesi. E. "The Human Engineering Task Area", Computer, Vol. 16, No. 11, November 1983, pp. 86-93.

Lindquist, T.E. "Assessing the Usability of Human-Computer Interfaces", IEEE Software, Vol. 2, No. 1, January 1985, pp. 74-82.

Mack, R.L., Lewis, C.H., and Carroll, J.M. "Learning to Use Word Processors: Problems and Prospects", ACM Transactions on Office Information Systems, Vol. 1, No. 3, July 1983, pp. 254-271.

Moran, T.P. "Guest Editor's Introduction: An Applied Psychology of the User", ACM Computing Surveys - Special Issue: The Psychology of Human-Computer Interaction, Vol. 13, No. 1, March 1981, pp. 1-11.

Nielsen, J. "Learning Difficulties of Programmers Faced with a New Programming Environment", Technical Report DAIMI PB-181, Computer Science Dept., Aarhus Univ., Aarhus, Denmark, 1984.

Reisner, P. "Formal Grammar and Human Factors Design of an Interactive Graphics System", IEEE Transactions on Software Engineering, Vol. SE-7, March 1981, pp.229-240.

Rubinstein, R. and Hersh, H. The Human Factor: Designing Computer Systems for People. Digital Press, 1984. Schneiderman, B. (1983) -To be added

Shackel, B. "The Concept of Usability", Distributed as part of a tutorial entitled "Usability: Its Meaning and Evaluation in Human-Computer Systems", Present at CHI'83: Human Factors in Computing Systems, December 1983.

Shneiderman, B. "Human Factors of Interactive Software," (In A. Blaser and M. Zoeppritz, Eds.) End User Systems and their Human Factors, Springer-Verlag Lecture Notes in Computer Science, Berlin, 1983, pp. 9-29.

Smith, S.L. and Aucella, A.F. "Design Guidelines for the User Interface to Computer-Based Information Systems", Tech. Report ESD-TR-83-122, USAF Electronic Systems Division, Hanscom Air Force Base, Mass., March 1983.

Yunten, T. and Hartson, H.R. "A SUPERvisory Methodology And Notation (SUPERMAN) for Human-Computer System Development", To appear in Advances in Human-Computer Interaction, Ablex Publishing Corp., May 1985.

Distribution List for IDA Paper P-1818

DoD

Col. Joe Greene — 10 copies
Director, STARS Joint Program Office
1211 Fern St., C-107
Arlington, VA 22202

Others

Defense Technical Information Center — 2 copies
Cameron Station
Alexandria, VA 22314

Dr. Elizabeth Bailey — 25 copies
400 N. Cherry St.
Falls Church, VA 22046

Rex Hartson
Department of Computer Science
562 McBryde Hall
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

Jakob Nielsen
IBM Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Ben Shneiderman
Computer Science Department
Unviersity of Maryland
College Park, Maryland 20742

CSED Review Panel

Dr. Dan Alpert, Director
Center for Advanced Study
University of Illinois
912 W. Illinois Street
Urbana, Illinois 61801

Dr. Barry W. Boehm
TRW Defense Systems Group, MS 2-2304
One Space Park
Redondo Beach, CA 90278

Dr. Ruth Davis
The Pymatuning Group, Inc.
2000 N. 15th Street, Suite 707
Arlington, VA 22201

Dr. Larry E. Druffel
Software Engineering Institute
Shadyside Place
480 South Aiken Av.
Pittsburgh, PA 15231

Dr. C.E. Hutchinson, Dean
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

Mr. A.J. Jordano
Manager, Systems & Software
Engineering Headquarters
Federal Systems Division
6600 Rockledge Dr.
Bethesda, MD 20817

Mr. Robert K. Lehto
Mainstay
302 Mill St.
Occoquan, VA 22125

Mr. Oliver Selfridge
45 Percy Road
Lexington, MA 02173

IDA

Gen. W.Y. Smith, HQ
Mr. Seymour Deitchman, HQ
Ms. Karen Webber, HQ
Dr. Jack Kramer, CSED
Dr. John Salasin, CSED
Dr. Robert Winner, CSED
Ms. Katydean Price, CSED 2 copies
IDA C&D Vault 3 copies

END

JAN.

1988

DTIC